МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Системи баз даних

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для здобувачів ступеня бакалавра за освітньою програмою "Комп'ютерний моніторинг та геометричне моделювання процесів і систем"

Київ
КПІ ім. Ігоря Сікорського
2019

Системи баз даних: Комп’ютерний практикум: навчальний посібник для здобувачів ступеня бакалавра за освітньою програмою "Комп'ютерний моніторинг та геометричне моделювання процесів і систем" / КПІ ім. Ігоря Сікорського; уклад.: І.В.Сегеда, О.А.Дацюк.– Електронні текстові дані (1 файл: 987 Кбайт). – Київ: КПІ ім. Ігоря Сікорського, 2019. – 43с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 7 від 01.04.2019)
за поданням Вченої ради теплоенергетичного факультету (протокол №8 від 25.03.2019)

Електронне мережне навчальне видання

Системи баз даних
КОМ’ЮТЕРНИЙ ПРАКТИКУМ

Укладачі: *Сегеда Ірина Василівна,*
канд. екон. наук, доцент
Дацюк Оксана Антонівна
Старший викладач

Відповідальний редактор: *Сидоренко Ю.В., канд. техн. наук, доцент*

Рецензент: *Тимошенко Ю.О., канд. техн. наук, доцент*

За редакцією укладача

Посібник розроблений на підставі робочої програми кредитного модуля «Системи баз даних» та призначений для якісної організації виконання комп’ютерного практикуму студентами.

Призначений для студентів бакалаврів, які навчаються за освітньою програмою "Комп'ютерний моніторинг та геометричне моделювання процесів і систем". Спрямований на формування у студентів умінь та досвіду роботи з реляційними базами даних засобами мови SQL. Забезпечує студентів теоретичними відомостями і необхідними прикладами виконання завдань, запланованих впродовж семестру.

ЗМІСТ

ВСТУП

Використання баз даних та інформаційних систем стає невід'ємною складовою діяльності сучасної людини та функціонування успішних організацій. У зв'язку з цим великої актуальності набуває опанування принципів побудови та ефективного застосування відповідних технологій та програмних продуктів: систем управління базами даних, систем автоматизації проектування.

Ефективність функціонування розроблених систем залежить від вірного вибору інструментальних засобів створення інформаційних систем, визначення відповідної моделі даних, обґрунтування раціональної схеми побудови баз даних, організації запитів до бази даних. Все це потребує усвідомленого застосування теоретичних положень та практичних навиків при розробці баз даних.

Мета циклу комп'ютерного практикуму полягає в тому, щоб студенти отримали навички у побудові та використанні баз даних, виконувати проектування інформаційного забезпечення (логічну та фізичну структури баз даних) інформаційних систем.

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №1.

Тема: **Робота в середовищі MySQL.**

Мета: Навчитися створювати і змінювати засобами SQL таблиці, задавати обмеження цілісності.

Завдання:

1) створити базу даних та таблицю за допомогою команд CREATE DATABASE та CREATE TABLE;
2) заповнити таблицю даними за допомогою команди INSERT
3) відредагувати дані в таблиці за допомогою команди UPDATE
4) видалити дані з таблиці командою DELETE
5) відредагувати структуру таблиці за допомогою команди ALTER TABLE
6) за допомогою команд редагування даних перевірити роботу встановлених правил цілісності даних (primary key, check, not null)
7) знищити таблицю та базу даних за допомогою команди DROP

ТЕОРЕТИЧНІ ВІДОМОСТІ

Мова SQL - перша і доки єдина стандартна мова для роботи з базами даних, яка отримала досить широке поширення. Практично усі видатні розробники систем керування базами даних (СКБД) створюють свої продукти з використанням мови SQL або з SQL -інтерфейсом. У нього зроблені величезні інвестиції як з боку розробників, так і з боку користувачів. Він став частиною архітектури додатків, є стратегічним вибором багатьох великих і впливових організацій.

Типи команд SQL

Реалізація в SQL концепції операцій, орієнтованих на табличне представлення даних, дозволила створити компактну мову з невеликим набором пропозицій. Мова SQL може використовуватися як для виконання запитів до даних, так і для побудови прикладних програм.

Основні категорії команд мови SQL призначені для виконання різних функцій, включаючи побудову об'єктів бази даних і маніпулювання ними, початкове завантаження даних в таблиці, оновлення і видалення існуючої інформації, виконання запитів до бази даних, керування доступом до неї і її загальне адміністрування.

Основні категорії команд мови SQL :

• DDL - мова визначення даних;

• DML - мова маніпулювання даними;

• DQL - мова запитів ;

• DCL - мова керування даними;

• команди адміністрування даних;

• команди керування транзакціями

Визначення структур бази даних (DDL)

Мова визначення даних (Data Definition Language, DDL) дозволяє створювати і змінювати структуру об'єктів бази даних, наприклад, створювати і видаляти таблиці. Основними командами мови DDL є наступні: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.

Маніпулювання даними (DML)

Мова маніпулювання даними (Data Manipulation Language, DML) використовується для маніпулювання інформацією усередині об'єктів реляційної бази даних за допомогою трьох основних команд: INSERT, UPDATE, DELETE.

Вибірка даних (DQL)

Мова запитів DQL найбільш відома користувачам реляційної бази даних, не дивлячись на те, що вона включає всього одну команду SELECT. Ця команда разом зі своїми численними опціями і пропозиціями використовується для формування запитів до реляційної бази даних.

Мова керування даними (DCL - Data Control Language)

Команди керування даними дозволяють управляти доступом до інформації, що знаходиться усередині бази даних. Як правило, вони використовуються для створення об'єктів, пов'язаних з доступом до даних, а також служать для контролю над розподілом привілеїв між користувачами. Команди керування даними наступні: GRANT, REVOKE.

Команди адміністрування даних

За допомогою команд адміністрування даних користувач здійснює контроль за виконуваними діями і аналізує операції бази даних; вони також можуть виявитися корисними при аналізі продуктивності системи. Не слід плутати адміністрування даних з адмініструванням бази даних, яке є загальним керуванням базою даних і має на увазі використання команд усіх рівнів.

Команди керування транзакціями

Існують наступні команди, що дозволяють керувати транзакціями бази даних: COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

Робота з таблицями:

1. Створення таблиць

Для створення таблиць програмним способом використовують оператор **CREATE TABLE**. Для цього потрібно вказати наступні дані:

- - ім'я таблиці, яке вказується після ключового слова **CREATE TABLE**;
- - імена та визначення стовпців таблиці, що відділені комами;
- - в деяких СКБД також вимагається, щоби було вказано місце розташування таблиці.

Створимо нову таблицю 1.1. та назвемо її **Customers**:

CREATE TABLE Customers (
ID CHAR (10) NOT NULL Primary key,
Custom_name CHAR(25) NOT NULL,
Custom_address CHAR(25) NULL,
Custom_city CHAR(25) NULL,
Custom_Country CHAR(25) NULL,
ArcDate CHAR(25) NOT NULL, DEFAULT NOWO)

Таблиця 1.1. **Customers**

ID	Custom_name	Custom_address	Custom_city	Custom_Country	ArcDate
*					

Так ми спочатку вказуємо назву нової таблиці, потім в дужках перераховуємо стовпці, які будемо створювати, причому їх назви не можуть повторюватися в межах однієї таблиці. Після назв стовпців вказується тип даних для кожного поля (**CHAR (10)**), потім зазначаємо чи може поле містити порожні значення (**NULL** або **NOT NULL**), а також потрібно вказати поле, яке буде первинним ключем (**Primary key**).

Мова SQL також дозволяє визначати для кожного поля значення по замовчуванню, тобто, якщо користувач не вкаже значення для певного поля - воно буде автоматично проставлене СКБД. Значення по замовчуванню визначається ключовим словом **DEFAULT** при визначенні стовпців оператором **CREATE TABLE**.

2. Оновлення таблиць

Для того, щоб змінити таблицю в SQL використовується оператор ALTER TABLE. При використанні даного оператора слід ввести наступну інформацію:

- - ім'я таблиці, яку ми хочемо змінити;
- - перелік змін, які ми хочемо зробити.

Для прикладу давайте додамо нову колонку в таблицю1.2. **Sellers**, в якій будемо зазначати телефон реалізатора:

ALTER TABLE Sellers ADD Phone CHAR (20)

Таблиця 1.2. **Sellers**

ID	Address	City	Seller_name	Country	Phone
1	500 Park Street	Montreal	Michelle Green	Canada	
2	1000 5th Avenue	San Francisco	Kim Howard	USA	
3	42 Galaxy Road	New York	John Smith	USA	
4	123 Main Street	Toronto	Denise L. Stephens	Canada	
5	4th Avenue	Ottawa	Semuel Piter	Canada	
6	1st Street	Los Angeles	Harry Monroe	USA	

Крім додавання стовпців, ми також можемо їх видаляти. Давайте тепер видалимо поле **Phone**. Для цього пропишемо наступний запит:

ALTER TABLE Sellers DROP COLUMN Phone

Таблиця1. 3. **Sellers**

ID	Address	City	Seller_name	Country
1	500 Park Street	Montreal	Michelle Green	Canada
2	1000 5th Avenue	San Francisco	Kim Howard	USA
3	42 Galaxy Road	New York	John Smith	USA
4	123 Main Street	Toronto	Denise L. Stephens	Canada
5	4th Avenue	Ottawa	Semuel Piter	Canada
6	1st Street	Los Angeles	Harry Monroe	USA

3. Видалення таблиць

Видалення таблиць здійснюється за допомогою оператора **DROP TABLE**. Щоби видалити таблицю **Sellers_new**, ми можемо прописати наступний запит:

DROP TABLE Sellers_new

В багатьох СКБД, застосовуються правила, що запобігають видаленню таблиць, які є вже пов'язані з іншими таблицями. Якщо ці правила діють і ви видаляєте таку таблицю, то СКБД блокує операцію видалення до тих пір, поки не буде видалений зв'язок. Такі заходи запобігають випадковому видаленню потрібних таблиць.

Доповнення даних

Для доповнення таблиці новими записами використовується команда INSERT.

Синтаксис команди такий:

INSERT [INTO] <ім`я таблиці> [(<список колонок>)] [DEFAULT] VALUES <список значень>

Команда дописує записи в кінець вказаної таблиці. VALUES вказує значення стовпчиків нового запису. Якщо не потрібно заповнювати всі поля нового запису, то у фразі INSERT після імені таблиці вказується список полів, які заповнюються значеннями, заданими у VALUES. Якщо імена полів опущені, то вказані вирази будуть записуватися послідовно в поля БД відповідно до її структури.

Наведемо приклади:

```
INSERT INTO customer (nom_pokup, name1, name2, name3, city)
        VALUES (2011, ‘Сидоров’, ‘Ілля’, ‘Петрович’, ‘Київ’)

INSERT INTO skalespeople
        VALUES (1001, ‘Іванов’, NULL, 12.5)
```

Фраза **DEFAULT** вказує на те, що добавлений запис буде містити значення за змовчанням, якщо воно визначене.

Наприклад,

```
INSERT INTO body  DEFAULT VALUES
```

Фраза DEFAULT зустрічається в команді INSERT INTO ще й в списку значень, коли при необхідності ним можна замінити значення, яке вводиться.

```
INSERT INTO body (name_body) VALUES (DEFAULT)
```

Команду INSERT можна використовувати разом з підзапитом, для того, щоб вибрати значення з однієї таблиці і перенести їх в іншу. Можна доповнювати запити з тієї ж таблиці. Для цього потрібно замінити VALUES на підзапит.

Наприклад:

```
INSERT INTO Londonstaff
                    SELECT *
                    FROM Salespeople
                    WHERE  cyti='London'

 INSERT INTO Daytotals  (date, total)
                SELECT zdate, SUM(amt)
                FROM zamovl
                GROUP BY zdate
```

(Відмітимо, що імена стовпчиків в двох таблицях не співпадають. Але якщо date і total єдині стовпчики таблиці і йдуть в ній в указаному порядку, то імена в INTO можна опустити.)

Зміна значень полів

По команді UPDATE можна змінювати деякі або всі значення в існуючій таблиці.

Синтаксис команди такий:

UPDATE <ім`я таблиці> SET <поле> = <значення> [, <поле> = <значення>...] [WHERE <умова>]

UPDATE замінює вказані поля таблиці новими значеннями.

SET вказує, які заміни потрібно виконати для вказаного стовпчика (стовпчиків).

Наприклад:

```
UPDATE Salespeople
   SET  sname='Іванов',  city='Мінськ'
   WHERE  snom=1004

UPDATE Salespeople
   SET  comm = comm*2
   WHERE   city = 'Мінськ'
```

В предикаті команди UPDATE можна використовувати підзапити.

Наприклад, підвищити комісійні для всіх продавців, які обслуговують більше одного покупця:

```
UPDATE Salespeople
   SET  comm = comm+0.02
   WHERE   2<=
                ( SELECT COUNT(nom_pokup)
                  FROM customer
                  WHERE customer.nom_prod= Salespeople.nom_prod )
```

Вилучення рядків з таблиці

Рядки з таблиці можна вилучити з допомогою команди DELETE. По цій команді вилучаються рядки повністю, а не окремі поля.

Синтаксис команди такий:

DELETE [FROM] <ім`я таблиці> [WHERE <умова>]

Наприклад:

```
DELETE  FROM customer
```
Якщо не вказані умови вибору, то вилучаються всі записи.

Найчастіше з таблиці потрібно вилучити тільки деякі рядки, ті, що відповідають заданій умові.

```
DELETE  FROM customer  WHERE city='Київ'
```

В предикаті команди DELETE можна використовувати підзапити. Це дає можливість формувати досить складні критерії вилучення рядків, це потребує особливої уваги і обережності.

Наприклад, якщо потрібно закрити лондонський офіс, то можна використати такий запит для вилучення всіх покупців цього офісу:

```
DELETE  FROM  customer
          WHERE snum=ANY
           (SELECT snum FROM Salespeople WHERE city='London')
```

Можна використовувати зв`язані підзапити. Наприклад, можна знайти найменше замовлення за кожний день і вилучити продавця, якому таке замовлення було адресовано.

```
DELETE  FROM Salespeople
          WHERE snum IN
                    (SELECT snum FROM Zamovl a
                     WHERE amt=
                              ( SELECT  MIN (amt)
                                FROM Zamovl b
                                WHERE a.zdate=b.zdate))
```

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №2.

Тема: **Створення БД. Аналіз функцій та побудова діаграми прецедентів в середовищі IBM Rational Rose**

Мета: Самостійно спроектувати БД по заданому опису проедметної області. Створити та заповнити відношення БД згідно свого варіанту. Визначити первинні ключі. Встановити зв`язки між таблицями. Задати обмеження цілісності.
Освоїти основні етапи проектування та особливості розробки діаграми діяльності в середовищі IBM Rational Rose.

Завдання:

1) створити БД, модель якої розроблено як індивідуальне завдання та перевірене викладачем;
2) створити таблиці спроектованої бази даних;
3) за допомогою візуальних засобів при редагуванні структури таблиці створити первинні та зовнішні ключі, та встановити необхідні правила на зв`язки між таблицями;
4) проаналізувати функції і побудувати відповідну діаграму прецедентів.

ТЕОРЕТИЧНІ ВІДОМОСТІ

РЕЛЯЦІЙНІ ЗВ'ЯЗКИ МІЖ ТАБЛИЦЯМИ БАЗ ДАНИХ

Зв'язки між об'єктами реального світу можуть знаходити своє віддзеркалення в структурі даних, а можуть і матися на увазі, тобто бути присутнім на неформальному рівні.

Між двома або більше таблицями бази даних можуть існувати відношення підлеглості, які визначають, що для кожного запису головної таблиці (званою ще батьківською) можлива наявність однієї або декількох записів в підпорядкованій таблиці (званою ще дочірньою).

Виділяють три різновиди зв'язку між таблицями бази даних :

• ""один-до-багатьох";

• ""один-до-одного";

• ""багато-до-багатьох".

Відношення "один-до-багатьох"

Відношення "один-до-багатьох" має місце, коли одному запису батьківської таблиці може відповідати декілька записів дочірньої. Зв'язок "один-до-багатьох" іноді називають зв'язком "багато-до-одного". І у тому, і в іншому випадку суть зв'язку між таблицями залишається незмінною. Зв'язок "один-до-багатьох" є найпоширенішим для реляційних баз даних. Він дозволяє моделювати також ієрархічні структури даних.

Відношення "один-до-одного"

Відношення "один-до-одного" має місце, коли одному запису у батьківській таблиці відповідає один запис в дочірній. Це відношення зустрічається набагато рідше, ніж відношення "один-до-багатьох". Його використовують, якщо не хочуть, щоб таблиця БД "розпухала" від другорядної інформації, проте для читання пов'язаної інформації в декількох таблицях доводиться робити ряд операцій читання замість однієї, коли дані зберігаються в одній таблиці.

Відношення "багато-до-багатьох"

Відношення "багато-до-багатьох" застосовується в наступних випадках:

• одному запису у батьківській таблиці відповідає більше ніж один запис в дочірній;

• одному запису в дочірній таблиці відповідає більше ніж один запис у батьківській.

Всякий зв'язок "багато-до-багатьох" в реляційній базі даних необхідно замінити на зв'язок "один-до-багатьох" (один або більше) за допомогою введення додаткових таблиць.

КЛЮЧІ

При роботі з таблицями в реляційних базах даних, бажано (необхідно), щоб кожна таблиця мала - первинний ключ.

Первинний ключ (primary key) – це поле, яке використовується для забезпечення унікальності даних в таблиці. Це означає, що значення (інформація) в полі первинного ключа в кожному рядку (запису) таблиці має бути унікальним.

Унікальність необхідна для уникнення неоднозначності, коли невідомо до якого запису таблиці потрібно звернутися, якщо в таблиці є записи що повторюються (два записи мають однакові значення у всіх полях таблиці).

Зовнішній ключ (foreign key)

– це одне або декілька полів (атрибутів), які є первинними в іншій таблиці і значення яких замінюється значеннями первинного ключа іншої таблиці.

Первинний, вторинний та зовнішній ключі можуть бути як простими так і складеними. **Прості ключі** – це ключі, що містять тільки одне поле (один атрибут). **Складені** (складні) **ключі** – це ключі, що містять декілька полів (атрибутів).

АНАЛІЗ ФУНКЦІЙ ТА ПОБУДОВА ДІАГРАМИ ПРЕЦЕДЕНТІВ В СЕРЕДОВИЩІ IBM RATIONAL ROSE

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (англ. *use case*)

використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою

Питання, що допоможуть визначити акторів та прецеденти наведены в таблиці 2.1.

Таблиця2.1.

Визначення акторів	Визначення прецедентів
Хто або що використовує систему? Які ролі вони граю у взаємодії? Хто встановлює систему? Хто або що запускає і вимикає систему? Хто обслуговує систему? Які системи взаємодіють з даною системою? Хто або що отримує і надає інформацію системі? Чи відбувається що-небудь в точно встановлений час? (Тоді в якості актора можна використовувати час)	Які функціональні можливості знадобляться конкретному актору від системи? Система зберігає і знаходить інформацію? Якщо так, який з акторів ініціює цю поведінку? Що відбувається, коли система змінює стан (наприклад, при запуску і виключенні системи)? Хто-небудь з акторів отримує при цьому повідомлення? Які небудь зовнішні події впливають на систему? Як система дізнається про ці події? Чи взаємодіє система з якою-небудь зовнішньою системою? Система генерує звіти?

Існує кілька ступенів формалізації прецедента:

1) стислий – анотація на один абзац, зазвичай вона описує лише основний успішний сценарій

2) вільний – неформальний стиль: опис займає кілька абзаців і охоплює кілька сценаріїв.

3) розгорнутий – найбільш докладний стиль опису, містить всі сценарії, передумови, результати,…

Використовуйте наступну **специфікацію опису прецедента**:

4) ім’я прецедента;

5) ID прецедента;

6) стислий опис;

7) актори, що задіяні в прецеденті (виконавці);

8) передумови (PD: Specification Tab Pre-Conditions);

9) постумови – результати, що отримують по закінченні прецедента (PD: Specification Tab Post-Conditions);

10) основний потік (PD: Specification Tab Action Steps);

11) альтернативні потоки або розширення (PD: Specification Tab Extension Points).

До цієї специфікації можна додати наступні параметри: зацікавлені актори, спеціальні вимоги, список технологій і типів даних, частота використання, відкриті питання.

Описувати основний сценарій за наступними правилами.

Описувати першу дію сценарію за шаблоном: «1. Прецедент починається, коли <актор> <дія>.»

Наступні кроки: <номер> <хто-небудь> <здійснює деяку дію>.

В розділі основного сценарію описуються три види дій:

1. Взаємодія між виконавцями (та системою)
2. Верифікація (зі сторони системи)
3. Зміна стану системи (наприклад запис або модифікація сутностей)

Для опису простого розгалуження можна використовувати ключове слово «Якщо», для опису дій, що повторюються можна використовувати ключове слово «Повторити Для» (For) <чого?>, а також «Виконувати Поки» (While).

Приклад

Основний потік:
1. Прецедент починається, коли Покупець вибирає опцію «знайти продукт».
2. Система запитує у Покупця критерій пошуку.
3. Покупець вводить запрошуваний критерій.
4. Система шукає продукти, відповідні критерію Покупця.
5. Якщо система знаходить відповідні продукти, тоді
    5.1. Для кожного знайденого продукту
        5.1.1. Система виводить на екран мініатюрне представлення продукту.
        5.1.2. Система виводить на екран короткий опис продукту.
        5.1.3. Система виводить на екран ціну продукту.
6. Інакше (Else)
6.1. Система повідомляє Покупця про те, що відповідні продукти не знайдені.

Альтернативні потоки можуть бути ініційовані трьома різними способами:

1. Альтернативний потік може бути ініційований *замість* основного потоку. Він ініціюється головним актором і повністю заміщає весь прецедент. Тобто його можна виділити в окремий прецедент.

2. Альтернативний потік може бути ініційований *після певного етапу* основного потоку. Тоді шаблон початку:

1. Альтернативний потік починається після кроку X основного потоку.

3. Альтернативний потік може бути ініційований *у будь-який момент* в ході виконання основного потоку (PD: Exceptions). Тоді шаблон початку:

1. Альтернативний потік починається у будь-який момент часу.

Такі альтернативні потоки використовуються для моделювання того, що може відбутися в будь-якій точці основного потоку до завершального етапу.

Щоб виявити альтернативні потоки, потрібно уважно вивчити основний потік. На кожному кроці основного потоку необхідно шукати:

- можливі альтернативи основному потоку;
- помилки, які можуть виникнути в основному потоці;
- переривання, які можуть трапитися в конкретній точці основного потоку;
- переривання, які можуть відбутися в будь-якій точці основного потоку.

Після деталізації прецедентів можна розширити діаграму прецедентів додатковими зв’язками.

Додаткові відношення на діаграмах прецедентів

Узагальнення акторів

Якщо кілька акторів мають схожу поведінку, для спрощення діаграми їх можна узагальнити. Це здійснюється шляхом створення нового абстрактного актора, котрий буде відображати ту частину поведінки акторів, що є однаковою для них.

Узагальнення прецедентів 

Узагальнення прецедентів виносить поведінку, загальну для одного або більше прецедентів, в батьківський прецедент. Нащадки можуть:

• успадковувати можливості батьківського прецеденту;

• вводити нові можливості;

• перевизначати (міняти) успадковані можливості.

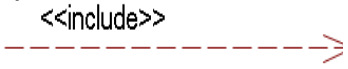
Дочірній прецедент автоматично успадковує всі можливості свого предка. Слід пам'ятати, що відношення та точки розширення прецеденту не можуть бути перевизначені.

Стандарту документування узагальнення прецедентів не існує, проте часто при цьому діють за наступними правилами:

Кожний номер кроку в нащадку супроводжується номером кроку відповідного батьківського прецеденту.

Якщо крок нащадка перевизначає крок батька, його номер супроводжується літерою «o» (overridden).

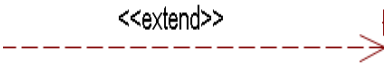
Якщо батьківський прецедент не містить потоку подій, або потік не завершено, він вважається абстрактним. Часто абстрактні прецеденти використовуються для найбільш загального опису поведінки системи, потік подій в них заміняє стислий опис. На практиці опис перевизначення та наслідування подій має ряд недоліків: необхідно використовувати додаткові теги, це ускладнює наочність моделі для замовника, при зміні батьківського прецеденту необхідно вручну змінювати нащадків. Застосування абстрактних прецедентів дозволяє цього уникнути.

Відношення «include» 

Іноді в різних прецедентах можуть бути описані однакові дії, тобто в прецедентах повторюються частини потоку подій. Для уникнення таких повторень використовується відношення «include».

Відношення «include» виносить кроки, загальні для кількох прецедентів, в окремий прецедент, який потім включається в інші.

Прецедент, що включає в себе інший прецедент називається *базовим*, а той, що включається – *включним*. В базовому прецеденті необхідно точно вказати місце, де повинна бути включена поведінка включного прецедента. На відповідному кроці сценарію пишеться include (<ім'я включного прецеденту>).

Відношення «extend» 

Надає можливість ввести нову поведінку в прецедент. Базовий прецедент надає набір *точок розширення*, для внесення нової поведінки. Розширюючий прецедент надає набір *сегментів вставки*, котрі можна ввести в місця, позначені точками розширення.

Точки розширення в основному потоці не нумерують, оскільки вони не мають впливати на основний потік. Основний потік є завершеним потоком і без розширень, він виступає як каркас для них. На жаль, часто розробники розуміють такий зв'язок по своєму, тому краще уникати частого використання відношення «extend».

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №3.

Тема: **Вибірка даних з створених відношень.**

Мета: Навчитися вибирати дані з однієї та декількох таблиць бази даних за допомогою оператора SELECT

Завдання:

1) створити запит по 1 таблиці за допомогою оператора SELECT:
   - для вибору стовпців таблиці;
   - WHERE для вибору рядків таблиці з використанням AND, OR, NOT;
   - WHERE з використанням предикатів IN, BETWEEN, LIKE, ISNULL;
2) створити впорядковані списки за допомогою ORDER BY;
3) створити запит по 2 таблицям з використанням INNER JOIN/ LEFT OUTER JOIN / RIGHT OUTER JOIN - пояснити різницю
4) створити запит з використанням опцій DISTINCT та LIMIT / TOP

ТЕОРЕТИЧНІ ВІДОМОСТІ

Найпершим та найголовнішим оператором в **SQL** являється ***SELECT***. З його допомогою ми можемо відбирати необхідні нам поля з даними в таблиці.

1. Вибірка

1.1. Вибірка окремих полів.

SELECT Product FROM Sumproduct в табл.3.1.

Таблиця 3.1. **Product**

Product
Bikes
Bikes
Bikes
Bikes
Skates
Skates
Skates
Skates
Skis Long
Skis Long

Бачимо, що наш **SQL запит** відібрав колонку **Product** з таблиці **Sumproduct**.

1.2. Вибірка кількох полів.

Припустимо, нам необхідно вибрати назву та кількість реалізованого товару. Для цього просто перераховуємо необхідні поля через кому:

SELECT Product, Quantity FROM Sumproduct в табл. 3.2.

Таблиця 3.2 **Sumproduct**

Product	Quantity
Bikes	12
Bikes	56
Bikes	854
Bikes	25
Skates	56
Skates	854
Skates	25
Skates	663
Skis Long	854
Skis Long	25
Skis Long	663

Після виразу SELECT (ім'я_стовпця) ...випливає FROM (ім'я_таблиці), а далі за допомогою конструкції WHERE може бути задана умова. Крім того, перед ім'ям стовпця може бути зазначено DISTINCT, і це означає, що враховуватися будуть тільки унікальні значення. За замовчуванням ж враховуються всі значення (для цього можна особливо вказати не DISTINCT, а ALL, але слово ALL не є обов'язковим).

1.3. Вибірка всіх стовпців.

Якщо ж нам необхідно отримати всю таблицю зі всіма полями, тоді просто ставимо знак зірочка (*):

SELECT * FROM Sumproduct в табл.3.3.

Таблиця 3.3. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
6	April	Skates	Montreal	56	$5 544,00
7	April	Skates	Toronto	854	$84 546,00
8	April	Skates	San Francisco	25	$2 475,00
9	April	Skates	New York	663	$65 637,00
10	April	Skis Long	Montreal	854	$209 230,00
11	April	Skis Long	Toronto	25	$6 125,00
12	April	Skis Long	San Francisco	663	$162 435,00
13	April	Skis Long	New York	21	$5 145,00
14	April	Skis Short	Montreal	21	$4 389,00
15	April	Skis Short	Toronto	4	$836,00
16	April	Skis Short	San Francisco	522	$109 098,00

Всі оператори в **SQL** нечутливі до регістру, тому ви можете їх писати як великими буквами, так і маленькими (як правило, їх прийнято писати великими буквами, щоб розрізняти від назв полів та таблиць). Назви же таблиць та полів є навпаки чутливими до регістру та мають писатися точно як в **БД**.

В майбутньому нам може знадобитися посортувати нашу вибірку - в алфавітному порядку для тексту чи по зростанню/спаданню - для цифрових значень. Для таких цілей в **SQL** є спеціальний оператор ***ORDER BY***.

2. Сортування

2.1. Сортування вибраних даних.

Давайте усю нашу таблицю посортуємо по сумі реалізації продукції, а саме по стовпцю **Amount**.

SELECT * FROM Sumproduct ORDER BY Amount Табл. 3.4.

Таблиця 3.4. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
47	January	Skates	New York	4	$396,00
15	April	Skis Short	Toronto	4	$836,00
35	February	Skis Short	Toronto	4	$836,00
74	March	Skis Short	Montreal	4	$836,00
52	January	Skis Long	San Francisco	4	$980,00
41	February	Snow Board	New York	4	$1 232,00
18	April	Snow Board	Montreal	4	$1 232,00
79	March	Snow Board	Toronto	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
28	February	Skates	San Francisco	21	$2 079,00
26	February	Skates	Montreal	21	$2 079,00
46	January	Skates	Montreal	21	$2 079,00
8	April	Skates	San Francisco	25	$2 475,00

Бачимо, що запит посортував записи у зростаючому порадку в полі **Amount**. Обов'язково потрібно дотримуватись послідовності розташування операторів, тобто оператор ***ORDER BY*** має йти в самому кінці запиту. В іншому випадку буде отримане повідомлення про помилку.

Також, особливістю оператора ***ORDER BY*** є те, що він може сортувати дані по полю, якого ми не вибирали у запиті, тобто достатньо щоб воно взагалі було в **БД**.

2.2. Сортування за кількома полями.

Тепер посортуємо наш приклад додатково за ще одним полем. Нехай це буде поле **City**, яке відображає місце реалізації продукції.

SELECT * FROM Sumproduct ORDER BY Amount, City Табл. 3.5.

Таблиця 3.5. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
47	January	Skates	New York	4	$396,00
74	March	Skis Short	Montreal	4	$836,00
35	February	Skis Short	Toronto	4	$836,00
15	April	Skis Short	Toronto	4	$836,00
52	January	Skis Long	San Francisco	4	$980,00
18	April	Snow Board	Montreal	4	$1 232,00
41	February	Snow Board	New York	4	$1 232,00
79	March	Snow Board	Toronto	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
26	February	Skates	Montreal	21	$2 079,00
46	January	Skates	Montreal	21	$2 079,00
28	February	Skates	San Francisco	21	$2 079,00

Черговість сортування буде залежати від порядку розташування полів в запиті. Тобто, в нашому випадку спочатку дані будуть посортовані по колонці **Amount**, а потім по **City**.

2.3. Напрямок сортування.

Незважаючи на те, що по замовчуванню оператор ***ORDER BY*** сортує по зростанню, ми можемо також прописати сортування значень по спаданню. Для цього в кінці кожного поля проставляємо оператор ***DESC*** (що є скороченням від слова ***DESCENDING).***

SELECT * FROM Sumproduct ORDER BY Amount DESC, City табл. 3.6.

Таблиця 3.6. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
4	April	Bikes	San Francisco	854	$320 250,00
22	February	Bikes	Montreal	663	$248 625,00
25	February	Bikes	New York	658	$246 750,00
70	March	Skis Long	Montreal	854	$209 230,00
10	April	Skis Long	Montreal	854	$209 230,00
21	April	Snow Board	New York	663	$204 204,00
59	January	Snow Board	Toronto	663	$204 204,00
63	March	Bikes	Montreal	522	$195 750,00
12	April	Skis Long	San Francisco	663	$162 435,00
32	February	Skis Long	San Francisco	663	$162 435,00
71	March	Skis Long	Toronto	663	$162 435,00
58	January	Snow Board	Montreal	522	$160 776,00
80	March	Snow Board	San Francisco	522	$160 776,00

В даному прикладі, значення в полі **Amount** були посортовані по спаданню, а в полі **City** - по зростанню. Оператор ***DESC*** застосовується лише для одного стовпця, тому при потребі його потрібно прописувати після кожного поля, яке приймає участь у сортуванні.

Найбільш потужною особливістю мови SQL є можливіть поєднувати різні таблиці в оперативній пам'яті СКБД під час виконання запитів. Об'єднання дуже часто використовуються для аналізу даних. Як правило, дані знаходяться в різних таблицях, що дозволяє їх більш ефективно зберігати (оскільки інформація не дублюється), спрощує обробку даних та дозволяє масштабувати базу даних (можливо додавати нові таблиці з додатковою інформацією).

3. об'єднання

3.1. Створення об'єднання таблиць

Об'єднання таблиць дуже проста процедура. Потрібно вказати всі таблиці, які будуть включені в об'єднання та "пояснити" СКБД , як вони будуть пов'язані між собою. Поєднання робиться за допомогою слова **WHERE**, наприклад:

```
SELECT DISTINCT Seller_name, Product
FROM Sellers, Sumproduct
WHERE Sellers.City = Sumproduct.City
```

табл. 3.7.

Таблиця 3.7. **Sumproduct.City**

Seller_name	Product
Denise L. Stephens	Bikes
Denise L. Stephens	Skates
Denise L. Stephens	Skis Long
Denise L. Stephens	Skis Short
Denise L. Stephens	Snow Board
John Smith	Bikes
John Smith	Skates
John Smith	Skis Long
John Smith	Skis Short
John Smith	Snow Board
Kim Howard	Bikes
Kim Howard	Skates
Kim Howard	Skis Long
Kim Howard	Skis Short
Kim Howard	Snow Board
Michelle Green	Bikes
Michelle Green	Skates
Michelle Green	Skis Long
Michelle Green	Skis Short
Michelle Green	Snow Board

Поєднавши дві таблиці, ми змогли побачити які товари реалізує кожен продавець. Розглянемо код запиту детальніше, оскільки він трохи відрізняється від звичайного запиту. Оператор **SELECT** починається з вказанням стовпців, які ми хочемо вивести, проте ці поля знаходяться в різних таблицях, речення **FROM** містить дві таблиці, які ми хочемо поєднати в операторі **SELECT**, таблиці поєднуються за допомогою слова **WHERE**, яке вказує стовпці для об'єднання. Обов'язково потрібно вказувати повну назву поля (*Таблиця.Поле*), оскільки поле **City** є в обох таблицях.

3.2. Внутрішнє об'єднання

В попередньому прикладі для об'єднання таблиць ми використали слово **WHERE**, яке здійснює перевірку на основі еквівалентності двох таблиць. Об'єднання такого типу називається також "*внутрішнім об'єднанням*". Існує також і інший спосіб об'єднання таблиць, який явно вказує на тип об'єднання. Розглянемо наступний приклад:

```
SELECT DISTINCT Seller_name, Product
FROM Sellers INNER JOIN Sumproduct ON Sellers.City = Sumproduct.City
```

табл.3.8.

Таблиця 3.8. Sumproduct.City

Seller_name	Product
Denise L. Stephens	Bikes
Denise L. Stephens	Skates
Denise L. Stephens	Skis Long
Denise L. Stephens	Skis Short
Denise L. Stephens	Snow Board
John Smith	Bikes
John Smith	Skates
John Smith	Skis Long
John Smith	Skis Short
John Smith	Snow Board
Kim Howard	Bikes
Kim Howard	Skates
Kim Howard	Skis Long
Kim Howard	Skis Short
Kim Howard	Snow Board
Michelle Green	Bikes
Michelle Green	Skates
Michelle Green	Skis Long
Michelle Green	Skis Short
Michelle Green	Snow Board

В цьому запиті замість **WHERE** ми використали конструкцію **INNER JOIN ... ON ...**, яка дала аналогічний результат. Незважаючи на те, що об'єднання з реченням **WHERE** є коротшим, краще використовувати **INNER JOIN**, оскільки вона є більш гнучкою, про що буде детальніше розказано в наступних розділах.

В більшості випадків необхідно отримувати не всі записи, а лише ті, які відповідають певним критеріям. Тому для здійснення фільтрації вибірки в **SQL** є спеціальний оператор ***WHERE***.

4.Фільтрування

4.1. Просте фільтрування оператором WHERE.

Давайте з нашої таблиці, наприклад, відберемо записи, які стосуються лише певного товару.Для цього ми зазначимо додатковий параметр відбору, який фільтруватиме значення по колонці **Product**.

Приклад запиту для відбору текстових значень:

SELECT *
FROM Sumproduct
WHERE **Product** = *'Bikes'* табл. 3.9.

Таблиця 3.9. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
22	February	Bikes	Montreal	663	$248 625,00
23	February	Bikes	San Francisco	21	$7 875,00
24	February	Bikes	San Francisco	54	$20 250,00
25	February	Bikes	New York	658	$246 750,00
42	January	Bikes	Montreal	75	$28 125,00
43	January	Bikes	Toronto	12	$4 500,00
44	January	Bikes	San Francisco	136	$51 000,00
45	January	Bikes	New York	21	$7 875,00
62	March	Bikes	Montreal	4	$1 500,00
63	March	Bikes	Montreal	522	$195 750,00
64	March	Bikes	San Francisco	125	$46 875,00
65	March	Bikes	New York	212	$79 500,00

Як бачимо, умова відбору взята в одинарні лапки, що є обов'язковим при фільтруванні текстових значень. При фільтруванні числових значень лапки не потрібні.

Приклад запиту для відбору числових значень:

SELECT * **FROM** **Sumproduct** **WHERE** **Amount** > *40000* **ORDER BY** **Amount**

В табл.3.10

Таблиця 3.10. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
39	February	Snow Board	Toronto	136	$41 888,00
61	January	Snow Board	New York	136	$41 888,00
64	March	Bikes	San Francisco	125	$46 875,00
44	January	Bikes	San Francisco	136	$51 000,00
48	January	Skates	San Francisco	522	$51 678,00
9	April	Skates	New York	663	$65 637,00
27	February	Skates	Toronto	663	$65 637,00
65	March	Bikes	New York	212	$79 500,00
7	April	Skates	Toronto	854	$84 546,00
67	March	Skates	Toronto	854	$84 546,00
36	February	Skis Short	San Francisco	522	$109 098,00
16	April	Skis Short	San Francisco	522	$109 098,00

В цьому прикладі ми відібрали записи, в яких виручка від реалізації становила більше **40 тис. $** та, додатково, всі записи посортували по зростанню по полю **Amount.**

В таблиці 3.11, зазначено перелік умовних операторів, які підтримуються **SQL**:

Таблиця 3.11.

Знак операції	Значення
=	Дорівнює
<>	Не дорівнює
<	Менше
<=	Менше або рівне
>	Більше
>=	Більше або рівне
BETWEEN	Між двома значеннями
IS NULL	Відсутній запис

4.2. Фільтрування по діапазону значень (BETWEEN).

Для відбору даних, які лежать в певному діапазоні, використовується оператор **BETWEEN**. В наступному запиті будуть відібрані усі значення, які лежать в межах від ***1000$*** до ***2000$*** включно, в полі **Amount**.

SELECT * **FROM** Sumproduct **WHERE** Amount **BETWEEN** *1000* **AND** *2000*

Таблиця 3.11. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
18	April	Snow Board	Montreal	4	$1 232,00
41	February	Snow Board	New York	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
79	March	Snow Board	Toronto	4	$1 232,00

Черговість сортування буде залежати від порядку розташування полів в запиті. Тобто, в нашому випадку спочатку дані будуть посортовані по колонці **Amount**, а потім по **City**.

4.3. Вибірка порожніх записів (IS NULL).

В **SQL** існує спеціальний оператор для вибірки порожніх записів (називаєьбся **NULL**). Порожнім записом вважається будь-яка комірка в таблиці, в яку не введено жодного символу. Якщо в комірку введено **0** або **пробіл**, то вважається, що поле заповнене.

SELECT * **FROM** Sumproduct **WHERE** Amount **IS NULL** табл. 3.13.

Таблиця 3.13. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
5	April	Bikes	New York	25	
19	April	Snow Board	Toronto	522	

В прикладі вище, ми навмисне видалили два значення в полі **Amount**, щоб продемонструвати роботу оператора **NULL**.

4.4. Розширене фільтрування (AND, OR).

Мова **SQL** не обмежується фільтруванням по одній умові, для власних цілей ви можете використовувати досить складні конструкції для вибірки даних одночасно по багатьом критеріям. Для цього в **SQL** є додаткові оператори, які розширюють можливості оператора **WHERE**. Такими операторами являються: **AND, OR, IN, NOT**. Наведемо кілька прикладів роботи даних операторів.

SELECT * **FROM** **Sumproduct** **WHERE** **Amount** > *40000* **AND** **City** = *'Toronto'*

Таблиця 3.14. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
7	April	Skates	Toronto	854	$84 546,00
19	April	Snow Board	Toronto	522	$160 776,00
27	February	Skates	Toronto	663	$65 637,00
39	February	Snow Board	Toronto	136	$41 888,00
59	January	Snow Board	Toronto	663	$204 204,00
67	March	Skates	Toronto	854	$84 546,00
71	March	Skis Long	Toronto	663	$162 435,00
75	March	Skis Short	Toronto	522	$109 098,00

SELECT * **FROM** **Sumproduct** **WHERE** **Month**= *'April'* **OR** **Month**= *'March'* в табл.3.15.

Таблиця 3.15. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
15	April	Skis Short	Toronto	4	$836,00
74	March	Skis Short	Montreal	4	$836,00
18	April	Snow Board	Montreal	4	$1 232,00
79	March	Snow Board	Toronto	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
8	April	Skates	San Francisco	25	$2 475,00
77	March	Skis Short	San Francisco	21	$4 389,00
14	April	Skis Short	Montreal	21	$4 389,00
2	April	Bikes	Montreal	12	$4 500,00
13	April	Skis Long	New York	21	$5 145,00
72	March	Skis Long	San Francisco	21	$5 145,00

Давайте об'єднаємо оператори **AND** та **OR**. Для цього зробимо вибірку велосипедів (***Bikes***) та ковзанів (***Skates***), які були продані в березні (***March***).

```
SELECT *
FROM Sumproduct
WHERE Product = 'Bikes' OR Product = 'Skates' AND Month= 'March' табл.
```
В табл.3.16.

Таблиця 3.16. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
22	February	Bikes	Montreal	663	$248 625,00
23	February	Bikes	San Francisco	21	$7 875,00
24	February	Bikes	San Francisco	54	$20 250,00
25	February	Bikes	New York	658	$246 750,00
42	January	Bikes	Montreal	75	$28 125,00
43	January	Bikes	Toronto	12	$4 500,00
44	January	Bikes	San Francisco	136	$51 000,00
45	January	Bikes	New York	21	$7 875,00
62	March	Bikes	Montreal	4	$1 500,00
63	March	Bikes	Montreal	522	$195 750,00
64	March	Bikes	San Francisco	125	$46 875,00
65	March	Bikes	New York	212	$79 500,00
66	March	Skates	Montreal	56	$5 544,00
67	March	Skates	Toronto	854	$84 546,00
68	March	Skates	San Francisco	212	$20 988,00
69	March	Skates	New York	56	$5 544,00

Бачимо, що в нашу вибірку попало за багато значень (крім березня (**March**), також січень (**January**), лютий (**February**) та квітень (**April**)). В чому ж причина? А в тому, що **SQL** має пріоритети виконання команд. Тобто оператор **AND** має вищий пріоритет, ніж оператор **OR**, тому спочатку було відібрано записи з ковзанами, що продані в березні, а потім усі записи, що стосувалися велосипедів.

Отже, щоб отримати правильну вибірку, нам потрібно змінити пріоритети виконання команд. Для цього використаємо **дужки**, як в математиці. Тоді, спочатку будуть опрацьовані оператори в дужках, а потім - всі решта.

```
SELECT * FROM Sumproduct WHERE (Product ='Bikes' OR Product = 'Skates')
AND Month= 'March'
```
в табл.3.17.

Таблиця 3.17. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
62	March	Bikes	Montreal	4	$1 500,00
63	March	Bikes	Montreal	522	$195 750,00
64	March	Bikes	San Francisco	125	$46 875,00
65	March	Bikes	New York	212	$79 500,00
66	March	Skates	Montreal	56	$5 544,00
67	March	Skates	Toronto	854	$84 546,00
68	March	Skates	San Francisco	212	$20 988,00
69	March	Skates	New York	56	$5 544,00

4.5. Розширене фільтрування (оператор IN).

SELECT * FROM Sumproduct WHERE ID IN (4, 12, 58, 67) табл. 3.18.

Таблиця 3.18. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
4	April	Bikes	San Francisco	854	$320 250,00
12	April	Skis Long	San Francisco	663	$162 435,00
58	January	Snow Board	Montreal	522	$160 776,00
67	March	Skates	Toronto	854	$84 546,00

Оператор **IN** виконує ту ж саму функцію, що і **OR**, проте має ряд переваг:

- при роботі з довгими списками, речення з **IN** легше читати;
- використовується менша кількість операторів, що пришвидшує обробку запиту;
- найважливіша перевага **IN** в тому, що в його конструкції можна використовувати додаткову конструкцію **SELECT**, що

відкриває великі можливості для створення складних підзапитів.

4.6. Розширене фільтрування (оператор NOT).

SELECT * FROM Sumproduct WHERE NOT City IN ('Toronto', 'Montreal') в табл.3.19.

Таблиця 3.19. **Sumproduct**

ID	Month	Product	City	Quantity	Amount
47	January	Skates	New York	4	$396,00
52	January	Skis Long	San Francisco	4	$980,00
41	February	Snow Board	New York	4	$1 232,00
28	February	Skates	San Francisco	21	$2 079,00
8	April	Skates	San Francisco	25	$2 475,00
77	March	Skis Short	San Francisco	21	$4 389,00
57	January	Skis Short	New York	21	$4 389,00
13	April	Skis Long	New York	21	$5 145,00
72	March	Skis Long	San Francisco	21	$5 145,00
33	February	Skis Long	New York	21	$5 145,00
69	March	Skates	New York	56	$5 544,00
40	February	Snow Board	San Francisco	21	$6 468,00

Ключове слово **NOT** дозволяє забрати непотрібні значення із вибірки. Також його особливістю є те, що воно проставляється перед назвою стовпця, який бере участь у фільтруванні, а не після.

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

Тема: **Вибірка даних з використанням розділів GROUP BY і HAVING**

Мета: Навчитися застосовувати агрегатні функції до груп записів, що мають спільні властивості

Завдання:

1) використати агрегатні функції MAX (), MIN (), AVG () та COUNT() без GROUP BY та із застосуванням GROUP BY;
2) пояснити дію розділу GROUP BY на прикладах із використанням та без використання агрегатних функцій;
3) створити запит з використанням розділу HAVING для завдання умов, що містять агрегатні функції та застосовуються після групування даних;
4) пояснити відмінність між розділами HAVING та WHERE у випадках коли проводимо та коли не проводимо групування даних
5) пояснити відмінність між розділами GROUP BY та ORDER BY, GROUP BY та DISTINCT.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Групування даних дозволяє розділити всі дані на логічні набори, завдяки чому стає можливим виконання статистичних обчислень окремо в кожній групі.

1. Створення груп (GROUP BY)

Групи створюються за допомогою речення **GROUP BY** оператора **SELECT**. Розглянемо на прикладі.

SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY Product табл.4.1.

Таблиця 4.1. **Product**

Product	Product_num
Bikes	3450
Skates	4376
Skis Long	5251
Skis Short	2879
Snow Board	3510

Даним запитом ми витягнули інформацію щодо кількості реалізованої продукції в кожному місяці. Оператор **SELECT** наказує вивести два стовпця **Product** - назва продукту та **Product_num** - розрахункове поле, яке ми створили для відображення кількості реалізованої продукції (формула поля **SUM(Quantity)**). Речення **GROUP BY** вказує СКБД згрупувати дані по стовпцю **Product**. Варто також зазначити, що **GROUP BY** має іти після речення **WHERE** та перед **ORDER BY**.

2. Фільтруючі групи (HAVING)

Так само, як ми фільтрували рядки в таблиці, ми можемо здійснювати фільтрацію по згрупованим даним. Для цього в **SQL** існує оператор **HAVING**. Візьмемо попередній приклад та додамо фільтрацію по групам.

```
SELECT Product, SUM(Quantity) AS Product_num  в табл. 4.2.
FROM Sumproduct
GROUP BY Product
HAVINGSUM(Quantity)>4000
```

Таблиця 4.2. **Product**

Product	Product_num
Skates	4376
Skis Long	5251

Бачимо, що після того, як була порахована кількість реалізованого товару в розрізі кожного продукту, СКБД "відсікла" ті продукти, яких було реалізовано менше 4000 шт.

Як бачимо, оператор **HAVING** дуже подібний до оператора **WHERE**, проте між собою вони мають суттєву відмінність: **WHERE** фільтрує дані до того, як вони будуть згруповані, а **HAVING** - здійснює фільтрацію після групування. Таким чином, рядки, які були вилучені реченям **WHERE** не будуть включені в групу. Отож, оператори **WHERE** та **HAVING** можуть використовуватись в одному реченні. Розглянемо приклад:

```
SELECT Product, SUM(Quantity) AS Product_num  в табл. 4.3.
FROM Sumproduct
WHERE Product<>'Skis Long'
GROUP BY Product
HAVING SUM(Quantity)>4000
```

Таблиця 4.3. **Product**

Product	Product_num
Skates	4376

Ми до попереднього прикладу добавили оператор **WHERE**, де вказали товар **Skis Long**, що в свою чергу вплинуло на групування оператором **HAVING**. Як результат бачимо, що товар **Skis Long** не попав в перелік груп з кількістю реалізованої продукції більше 4000 шт.

3. Групування та сортування

Як і при звичайній вибірці даних, ми можемо посортувати групи після групування оператором **HAVING**. Для цього ми можемо використати вже знайомий нам оператор **ORDER BY**. В даній ситуації його застосування аналогічне попереднім прикладам. Наприклад:

SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY Product HAVINGSUM(Quantity)>3000 ORDER BY SUM(Quantity) або просто вкажемо номер поля по порядку, по якому хочемо посортувати:

SELECT Product, SUM(Quantity) **AS** Product_num **FROM** Sumproduct **GROUP BY** Product **HAVING**SUM(Quantity)>**3000 ORDER BY 2** в табл. 4.4.

Таблиця 4.4. **Product**

Product	Product_num
Bikes	3450
Snow Board	3510
Skates	4376
Skis Long	5251

Бачимо, що для сортування зведених результатів нам потрібно просто прописати речення з **ORDER BY**після оператора **HAVING**.

Агрегатні функції SQL діють відносно значень стовпця з метою отримання єдиного результуючого значення. Найбільш часто застосовуються такі агрегатні функції SUM, MIN, MAX, AVG і COUNT. Слід розрізняти два випадки застосування агрегатних функцій. Перший: агрегатні функції використовуються самі по собі і повертають одне результуюче значення. Другий: агрегатні функції використовуються з оператором SQL GROUP BY, тобто з групованням по полях (стовпцях) для отримання результуючих значень в кожній групі.

Функція SUM повертає суму значень стовпця таблиці бази даних. Вона може застосовуватися тільки до стовпців, значеннями яких є числа. Запити SQL для отримання результуючої суми починаються так:

SELECT SUM (ім'я_стовпця) ...

Після цього виразу випливає FROM (ім'я_таблиці), а далі за допомогою конструкції WHERE може бути задана умова.

Функція MIN також діє щодо стовпців, значеннями яких є числа і повертає мінімальне серед всіх значень стовпця. Ця функція має синтаксис аналогічний синтаксису функції SUM.

Функція MAX аналогічно працює і має аналогічний синтаксис функція SQL MAX, яка застосовується, коли потрібно визначити максимальне значення серед всіх значень стовпця.

Функція AVG- зазначене щодо синтаксису для попередніх описаних функцій вірно і щодо функції SQL AVG. Ця функція повертає середнє значення серед всіх значень стовпця.

Функція COUNT повертає кількість записів таблиці бази даних. Якщо в запиті вказати SELECT COUNT (ім'я_стовпця) ..., то результатом буде кількість записів без урахування тих записів, в яких значенням стовпця є NULL (невизначений). Якщо використовувати в якості аргументу зірочку і почати запит SELECT COUNT (*) ..., то результатом буде кількість всіх записів (рядків) таблиці.

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №5

Тема: **Використання виразу CASE у вибірках даних. Оператор UNION.**

Мета: Навчитися застосовувати умовний оператор в команді SELECT

Завдання:

1) написати пошуковий запит SELECT з використанням двох типів виразу CASE (простий вираз / логічний вираз);
2) написати запит з використанням інструкції IIF / IF (MS SQL / MySQL);
3) написати запит SELECT з використанням CASE або IIF в різних розділах пошукового запиту (SELECT, IN, WHERE, ORDER BY, HAVING)
4) навести приклади використання CASE або IIF в інших запитах (UPDATE, DELETE)
5) написати запит з використанням функції CHOOSE / COALESCE (MS SQL / MySQL);
6) Написати запит з використанням оператора UNION. Пояснити відмінність UNION від JOIN.
7) створити запит з використанням розділу HAVING для завдання умов, що містять агрегатні функції та застосовуються після групування даних;
8) пояснити відмінність між розділами HAVING та WHERE у випадках коли проводимо та коли не проводимо групування даних
9) пояснити відмінність між розділами GROUP BY та ORDER BY, GROUP BY та DISTINCT.

ТЕОРЕТИЧНІ ВІДОМОСТІ

В більшості **SQL-запитів** використовується один оператор, за допомогою якого повертаються дані із однієї або кількох таблиць. **SQL** також дозволяє виконувати одночасно кілька окремих запитів та відображати результат у вигляді єдиного набору даних. Такі комбіновані запити зазвичай називають *поєднаннями* або *складними запитами*.

1. Використання оператора UNION

Запити в мові **SQL** комбінуються за допомогою оператора **UNION**. Для цього необхідно вказати кожен запит **SELECT** та розмістити між ними ключове слово **UNION**. Обмежень щодо кількості використанньо оператора **UNION** в одному загальному запиті немає.

```
SELECT *
FROM Sumproduct LEFT JOIN Sellers ON Sumproduct.City = Sellers.City
UNION
SELECT *
FROM Sumproduct RIGHT JOIN Sellers ON Sumproduct.City = Sellers.City
```

В табл. 5.1.

Таблиця 5.1.

Sumproduct.ID	Month	Product	Sumproduct	Quantity	Amount	Sellers.ID	Address	Sellers.City	Seller_name	Country
						5	4th Avenue	Ottawa	Semuel Piter	Canada
2	April	Bikes	Montreal	12	4 500,00 грн.	1	500 Park Street	Montreal	Michelle Greei	Canada
3	April	Bikes	Montreal	56	21 000,00 грн.	1	500 Park Street	Montreal	Michelle Greei	Canada
4	April	Bikes	San Francisco	854	320 250,00 грн.	2	1000 5th Avenue	San Francisco	Kim Howard	USA
5	April	Bikes	New York	25	9 375,00 грн.	3	42 Galaxy Road	New York	John Smith	USA
6	April	Skates	Montreal	56	5 544,00 грн.	1	500 Park Street	Montreal	Michelle Greei	Canada
7	April	Skates	Toronto	854	84 546,00 грн.	4	123 Main Street	Toronto	Denise L. Stepl	Canada
8	April	Skates	San Francisco	25	2 475,00 грн.	2	1000 5th Avenue	San Francisco	Kim Howard	USA
9	April	Skates	New York	663	65 637,00 грн.	3	42 Galaxy Road	New York	John Smith	USA
10	April	Skis Long	Montreal	854	209 230,00 грн.	1	500 Park Street	Montreal	Michelle Greei	Canada

Бачимо, що запит відобразив як всі колонки з першої таблиці - так і з другої, незалежно від того, чи всі записи мають відповідники у іншій таблиці.

Також варто зазначити, що в багатьох випадках замість **UNION** ми можемо використовувати речення **WHERE** з багатьма умовами, та отримувати аналогічний результат. Проте через **UNION** записи виглядають більш лаконічними та зрозумілими. Також необхідно дотримуватись певних правил при написанні комбінованих запитів:

- запит **UNION** повинен включати два і більше операторів **SELECT**, відділених між собою ключовим словом **UNION** (тобто якщо в запиті використовується чотири оператора SELECT, то повинно бути три ключових слова UNION);
- кожен запит в операторі **UNION** повинен мати одні й ті ж стовпці, вирази чи статистичні функції, які, до того ж, мають бути перераховані в однаковому порядку;
- типи даних стовпців мають бути сумісними. Вони не обов'язково мають бути одного типу, проте мають мати подібний тип, щоби **СКБД** могла їх однозначно перетворити (наприклад, це можуть бути різні числові типи даних або різні типи дати).

2. Включення або виключення повторюваних рядків

Запит з **UNION** автоматично видаляє усі повторювані рядки з набору результатів запиту (іншими словами, веде себе як речення **WHERE** з кількома умовами в одному операторі **SELECT**). Така поведінка оператора **UNION** по замовчуванню, але при бажанні ми можемо змінити це. Для цього нам варто використовувати оператор **UNION ALL** замість **UNION**.

3. Сортування результатів комбінованих запитів

Результати виконання оператора **SELECT** сортуються за допомогою речення **ORDER BY**. При комбінуванні запитів за допомогою **UNION** тільки одне речення **ORDER BY** може бути використане, і воно має бути проставлене в останньому операторі **SELECT**. Дійсно, на практиці немає особливого змісту частину результатів сортувати в одному порядку, а іншу частину - в іншому. Тому кілька речень **ORDER BY**застосовувати не дозволяється.

4. Використання оператора CASE

CASE – умовний оператор мови SQL. CASE дозволяє провести перевірку умови та, в залежності від результату перевірки, повертає результат.

Оператор CASE має 2 форми:

```
CASE
WHEN умова_1
THEN результат_1
…
WHEN умова _N
THEN результат _N
[ELSE результат ]
END
```

```
CASE  значення для перевірки
WHEN  значення для порівняння_1
THEN   результат _1
…
WHEN  значення для порівняння _N
THEN   результат _N
[ELSE    результат]
END
```

```
SELECT  ID_tov, Nazv, Price, Price*0,9
  CASE WHEN Price>500 THEN Price*0,9 END,
  CASE WHEN Price<=500 THEN Price END,
FROM Tovar

SELECT  Prizv, kurs=
  CASE kurs
      WHEN 1 THEN ‘перший’,
      WHEN 2 THEN ‘другий’,
      WHEN 3 THEN ‘третій’,
  END
FROM Stud

SELECT  kurs, CASE
      WHEN kurs=1 THEN ‘абітурієнти’,
      WHEN kurs IN (2,3,4) THEN ‘бакалаври’,
      WHEN kurs IN (5,6)   THEN
                        CASE PositionID
                            WHEN 6 THEN ‘спеціалісти’,
                            WHEN 7 THEN ‘магістри’,
                        END,
      END  riven,
  COUNT(*) Cnt
FROM Stud
GROUP BY  kurs

SELECT
  kurs,
```

```
  SUM(CASE WHEN kurs =1 THEN Stip END),
  SUM(CASE WHEN kurs =2 THEN Stip END),
  SUM(CASE WHEN kurs =3 THEN Stip END),
  SUM(CASE WHEN kurs =4 THEN Stip END),
  SUM(Salary) Sum_bakalavr
FROM Stud
Where kurs<=4
GROUP BY kurs


SELECT
  kurs,
  SUM(IIF(kurs =1, Stip, NULL)),
  SUM(IIF(kurs =2, Stip, NULL)),
  SUM(IIF(kurs =3, Stip, NULL)),
  SUM(IIF(kurs =4, Stip, NULL)),
  SUM(Salary) Sum_bakalavr
FROM Stud
Where kurs<=4
GROUP BY kurs


SELECT  ID,
  CASE
    WHEN YEAR(Data_v)>=2000 THEN 'XXI'
    WHEN YEAR(Data_v)>=1900 THEN 'XX'
    WHEN Data_v IS NOT NULL THEN 'давні'
    ELSE 'не вказано'
  END
  COUNT(*) EmplCount
FROM VVV
GROUP BY
CASE
    WHEN YEAR(Data_v)>=2000 THEN 'XXI'
    WHEN YEAR(Data_v)>=1900 THEN 'XX'
    WHEN Data_v IS NOT NULL THEN 'давні'
    ELSE 'не вказано'
  END
```

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №6

Тема: **Вивчення основ реляційної алгебри (РА). Ознайомлення з основними принципами нормалізації таблиць у Реляційній моделі бази даних.**

Мета: Вивчення основних і додаткових операцій реляційної алгебри.
Формуваня практичних навичок нормалізації відношень в реляційних моделях даних.

Завдання:

1. Необхідно написати на мові SQL запити, які реалізують операції реляційної алгебри. Якщо для демонстрації операцій РА недостатньо відношень, створених під час виконання роботи №2, то слід створити додаткові відношення.
2. Перевірити розроблену бд на відповідність вімог 3 нф.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Оператор декартового добутку

Реляційна алгебра: A Times B
Оператор SQL: SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ...FROM A, B;
або SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ...FROM A CROSS JOIN B;

Оператор проекції

Реляційна алгебра: A [X,Y,..,Z]
Оператор SQL:
```
SELECT DISTINCT X, Y, ..., Z
FROM A;
```

Оператор вибірки

Реляційна алгебра: A Where C,
Оператор SQL:
```
SELECT *
FROM A
WHERE C;
```

Оператор об'єднання

Реляційна алгебра: A Union B
Оператор SQL:
```
SELECT *
FROM A
UNION SELECT *
FROM B;
```

Оператор віднімання

Реляційна алгебра:A Minus B
Оператор SQL:
```
SELECT *
FROM A
EXCEPT SELECT * FROM B
```

Реляційний оператор перейменування RENAME виражається за допомогою ключового слова AS у переліку полів оператора, які відбираються, SELECT. Таким чином, мова SQL є реляційно повною.

Інші оператори реляційної алгебри (з'єднання, перетинання, розподіл) виражаються через примітивні, отже, можуть бути виражені операторами SQL. Проте, для практичних цілей наведемо їх.

Оператор з'єднання / сполучення

Реляційна алгебра: (A Times B) Where C

Оператор SQL: SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ...FROM A, B WHERE C;

або SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ... FROM A CROSS JOIN B

WHERE C;

Оператор перетинану

Реляційна алгебра: A Intersect B

Оператор SQL: SELECT *
FROM A
INTERSECT SELECT * FROM B;

Оператор розподілу/ ділення

Реляційна алгебра: A (X,Y) Devid By B(Y)

Оператор SQL: SELECT DISTINCT A.X
FROM A
WHERE NOT EXIST (SELECT * FROM B WHERE NOT EXIST
(SELECT * FROM A A1
WHERE A1.X = A.X AND
A1.Y = B.Y));

Нормалізація відношень баз даних. Надлишковість даних і аномалії оновлення.

Нормалізація відношень - покроковий процес розділення (декомпозиції) початкових відношень БД на простіші. Кроки цього процесу переводять схему відношення БД в послідовні ***нормальні форми***. Кожна наступна форма володіє кращими властивостями ніж попередня. Кожній нормальній формі відповідає певний набір обмежень. При переведенні структури відношення у форми вищого порядку досягають видалення з таблиць надмірної описової інформації. Процес нормалізації заснований на понятті функціональної залежності атрибутів.

Функціональна залежність описує зв'язок між атрибутами таблиці. Якщо в таблиці N, що містить атрибути A и B, атрибут У функціонально залежить від атрибута A (A(B), то кожне значення атрибута A зв'язано тільки з одним значенням атрибута B.

Детермінант функціональної залежності – атрибут, від якого залежить інший атрибут цієї ж таблиці.

Перша нормальна форма. Відношення відповідає 1*NF* тоді, коли на перетині кожного стовпця і кожного рядка знаходяться тільки елементарні (неподільні) значення атрибутів і не містяться групи, що повторюються.

Друга нормальна форма.Відношення знаходиться в 2*NF*, якщо виконуються обмеження 1*NF* і кожен описовий атрибут функціонально повно залежить від первинного ключа (у тому числі і складеного).

Третя нормальна форма.Відношення знаходиться у 3*NF*, якщо виконуються обмеження 2*NF* і всі описові атрибути відношення взаємно незалежні і повністю

залежать від первинного ключа, тобто кожний описовий атрибут не транзитивно залежить від ключа.

Четверта нормальна форма (4НФ, 4NF) вимагає, аби в схемі баз даних не було нетривіальних багатозначних залежностей множин атрибутів від будь чого, окрім надмножини ключа-кандидата. Вважається, що таблиця знаходиться у 4НФ тоді, і тільки тоді, коли вона знаходиться в НФБК, та багатозначні залежності є функціональними залежностями. Четверта нормальна форма усуває небажані структури даних — багатозначні залежності.

П'ята нормальна форма (5НФ, 5NF, PJ/NF) вимагає, аби не було не тривіальних залежностей об'єднання, котрі б не витікали із обмежень ключів. Вважається, що таблиця в п'ятій нормальній формі, тоді, і тільки тоді, коли вона знаходиться в 4НФ, та кожна залежність об'єднання зумовлена її ключами-кандидатами.

Цілі нормалізації наступні: Виключити дублювання інформації в таблицях. Забезпечити можливість змін у структурі таблиць. Зменшити вплив структурних змін бази даних на роботу додатків, які забезпечують користувачам доступ до даних.

Аномаліями називатимемо таку ситуацію в таблицях БД, яка приводить до суперечностей в БД або істотно ускладнює обробку даних.

Виділяють три основні види аномалій: аномалії модифікації (або редагування, оновлення), аномалії видалення і аномалії додавання (або вставки).

Аномалії модифікації (UPDATE) виявляються в тому, що зміна значення одного даного може спричинити проглядання всієї таблиці і відповідну зміну деяких інших записів таблиці.

Аномалії видалення (DELETE) полягають в тому, що при видаленні якого-небудь даного з таблиці може пропасти і інша інформація, яка не пов'язана безпосередньо з даним, що видаляються.

Аномалії додавання (INSERT) виникають у випадках, коли інформацію в таблицю не можна помістити до тих пір, поки вона неповна, або вставка нового запису вимагає додаткового перегляду таблиці.

Другим прикладом виникнення аномалії додавання може бути ситуація включення в таблицю нового співробітника. При додаванні таких записів для виключення суперечностей бажано перевірити номер телефону і відповідний номер кімнати хоч би з одним із співробітників, що сидять з новим співробітником в тій же кімнаті. Якщо ж станеться, що у декількох співробітників, що сидять в одній кімнаті, є різні телефони, то взагалі не ясно, що робити (чи то в кімнаті декілька телефонів, чи то якийсь з номерів помилковий).

Детальний розгляд процесу нормалізації та функціональні залежності – рекомендована література п. №1.

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №7

Тема: **Технічне завдання (ТЗ)**

Мета: Навчитися самостійно писати технічне завдання для свого проекту.

Завдання:

1) Ознайомитись з гостами для написання ТЗ:
   - ГОСТ 19.201-78 Єдина система програмної документації. Технічне завдання. Вимоги до змісту та оформлення.
   - ГОСТ 2.114-95 Єдина система конструкторської документації. Технічні умови
   - ГОСТ 34.602-89 Інформаційна технологія. Комплекс стандартів на автоматизовані системи. Технічне завдання на створення автоматизованої системи.
2) Написати ТЗ для проекту згідно з заданою предметною областю. ТЗ роздрукувати. Приклад технічного завдання наведено в додатку №1.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Що таке програмна документація — Software Documentation?

Проектна документація — описує програмний продукт, як правило, у загальних рисах. Які патерни застосовуватимуться. Як конструюватимуться класи. Чому структура даних організована саме таким чином, а не інакше. Часто даються вказівки, як згодом слід модернізувати програму.

Технічне Завдання — це частина проектної документації. У Технічному Завданні вказується призначення програмного продукту, характеристики, а також конкретні методи реалізації.

Технічне Завдання — це юридичний документ, котрий у якості додатку включається в основний договір із замовником. Такий договір підписується перед початком робіт. Хоча інколи трапляється, коли ТЗ узагалі єдиний документ до проекту. Тому у таких випадках важливо щоби ТЗ розроблялося професіоналом із докладною деталізацією.

Технічна Документація — це документ який формується по ходу роботи, а остаточно впорядковується уже на фініші. У Технічній документації конкретно вже перелічують код, алгоритм, структури, API — з паралельним описом, що вони роблять.

Технічну Документацію зручно складати за допомогою спеціальних сервісів генераторів документації: **(Doxygen, NDoc, Javadoc, Visual Expert, EiffelStudio, Sandcastle, ROBODoc, POD, TwinText, Universal Report ін.)** Генератори документації дуже зручно допомагають постійно підтримувати документацію в актуальному стані. Ці інструменти витягують коментарі із коду. За потреби зміст можна редагувати. Плюс їх можна ще використовувати при збірках програми.

Отже, Технічна Документація є складовою частиною вихідного коду. Файл **README** помічали? Це воно!

Також Технічна документація може бути сформована у текстові довідкові посібники до друку, кому як зручно.

Технічна Документація використовується усіма: розробниками, тестувальниками, адміністраторами, кінцевими користувачами, які використовують програмне забезпечення. Для усіх вона є важливою, бо у ній фіксуються усі зміни у програмі.

Створює Технічну документацію програміст. **Technical Writer** формалізує її до «дружньої» форми. Завдання тестера відповідно протестувати-звірити на кожному етапі процесу тестування чи програма виконує задекларованe у Технічній Документації. З хорошою документацією тестеру тестувати досить легко і приємно. Такий підхід, як мінімум, запевняє у максимальному тестовому покритті проекту.

Загалом принципи до створення Технічної Документації ті ж самі що й до коментування коду:

- чіткість,
- структурованість,
- логічність,
- однозначність,
- зрозумілість,
- стисліcть,
- повнота вмісту інформації.

І своєчасність — якщо явно змінили поведінку у програмі, будьте добрі відобразити це коментарями й оновити документацію.

Усі текстові документи укладаються за міжнародними стандартами ISO переважно охоплюються стандартом ICS 01.110, кодові хто як хоче, але гарно би було у форматах: (DITA), DocBook, S1000D.

Хороша для користувача Технічна Документація складається з:

- вступного слова, де розглядаються загальні завдання програми і які проблеми вона вирішує;
- інформацію про інтелектуальну власність, авторське право, патент;
- тематичного керівництва, де кожна глава присвячена роз’ясненню певного розділу експлуатації програми;
- алфавітного показчика, для досвідчених користувачів, які добре знають, що їм потрібно шукати у програмі.

Рекомендована література:

1. Дейт, К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ / Дейт, К. Дж. – М.: Издательский дом «Вильямс», 2005. – 1328 с. – (8-е издание)
2. Берко А. Ю. Системи баз даних та знань: навч. посіб. / А. Ю. Берко, О. М. Верес, В. В. Пасічник; - Львів : Магнолія, 2008. - 456 с. - Мін-во освіти і науки України. - (Серія «Комп'ютинг» / за заг. ред. В. В. Пасічника; [кн. 1]).
3. Фиайли К. SQL: Руководство по изучению языка [Електронний ресурс] / Крис Фиайли. – 2013. – Режим доступу до ресурсу: http://smotrim.by/books/4221-sql-rukovodstvo-po-izucheniyu-yazyka-kris-fiayli-2013.html).
4. Уроки SQL [Електронний ресурс] - Режим доступу до ресурсу: http://moonexcel.com.ua/уроки-sql.
5. Управляючі Конструкції sql. [Електронний ресурс] - Режим доступу до ресурсу: https://studfiles.net/preview/5210288/page:2/)
6. Учебник по языку SQL (DDL, DML) на примере диалекта MS SQL Server. Часть третья. [Електронний ресурс] – Режим доступу до ресурсу: https://habrahabr.ru/post/255825/).
7. Організація баз даних та знань. Реляційна алгебра. [Електронний ресурс] – Режим доступу до ресурсу: http://bookwu.net/book_organizaciya-baz-danih-i-znan_997/27_2.10-realizaciya-relyacijno-algebri.
8. Що таке технічне завдання? [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/shho-take-tehnichna-dokumentatsiya-ta-pryntsypy-yiyi-stvorennya/
9. Буч Г., Якобсон А., Рамбо Дж. UML. Классика компьютерных технологий: Пер. с англ. – СПб.: Питер, 2006. – 736 с.
10. Леоненков А.В. Самоучитель UML. / А.В. Леоненков– СПб.: БХВ-Петербург, 2006. – 432 с.
11. Форта Б. Освой самостоятельно SQL. 10 минут на урок, 3-е издание: пер. с англ.. / Бен Форта. – М: Издательский дом “Вильямс”, 2006. – 288 с.

Додаток №1

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра АПЕПС

ТЕХНІЧНЕ ЗАВДАННЯ

по темі:

листів

Консультант по спеціальній частині:

_____ ФІО

Керівник роботи::

_____

Дата видачі завдання:

«____» ____ 201__ г.

Завдання прийнято до виконання:
студент групи ТМ-
_____ ФІО

Нормоконтроль:

_____

КИЇВ -2019

1. ВСТУП

В даний час…

Тому сьогодні актуальним є вирішення завдання ...

Для досягнення цієї мети необхідно ...

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Наказ по університету від "___" _____ 201_ р за № ____.

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Автоматизована інформаційна система (АІС) призначена для ...

У базі даних АІС повинна зберігатися інформація ...

4. ВИМОГИ ДО ПРОГРАМИ

4.1.Вимоги до функціональних характеристик

АІС повинна виконувати наступні функції:

• зберігання інформації ... (ВКАЗАТИ, ЯКУ ІНФОРМАЦІЮ);

• реєстрація користувачів, перевірка прав доступу;

• забезпечення несуперечності даних;

• пошук даних відповідно до призначених для користувача запитів;

• аналіз даних ... (ВКАЗАТИ, ЯКИХ ДАНИХ І ЗА ЯКИМИ КРИТЕРІЯМ);

• отримання звітів ... (ВКАЗАТИ, ЯКИХ САМЕ ЗВІТІВ І з якою періодичністю, ЯКЩО ЦЕ ВИЗНАЧЕНО).

Вхідні дані для створення бази даних - опис предметної області (або файли формату ... або документи ...).

Вихідні дані: документи заданих форматів, що виводяться на дисплей, в файл або на принтер; результати виконання запитів; ...

Вимоги до тимчасових характеристик АІС: час обробки запиту на отримання інформації ... не повинно перевищувати ... (або: вимог до тимчасових характеристик не пред'являється).

Вимоги до ємнісних характеристик: обсяг програмного коду не повинен перевищувати ... (або: вимог до ємнісних характеристик не пред'являється).

4.2.Вимоги до надійності

База даних повинна здійснювати контроль обмежень цілісності, обумовлених предметною областю.

АІС повинна працювати з базою даних розробленої конфігурації відповідно до алгоритму функціонування.

Для запобігання некоректної роботи програми необхідно реалізувати:

• авторизацію доступу до даних;

• семантичний і синтаксичний контроль вихідних даних;

• вивід повідомлень про помилки;

• можливість повторного введення даних.

Збереження даних повинно забезпечуватися засобами системи управління базами даних (СКБД). Для захисту від втрати даних необхідно розробити стратегію резервного копіювання.

4.3.Умови експлуатації

Умови експлуатації АІС (бази даних) збігаються з умовами експлуатації ПЕОМ IBM PC і сумісних з ними ПК (для розподіленої системи або системи, що працює в режимі клієнт-сервер - додати фразу "і мережевого устаткування").

4.4.Вимоги до складу і параметрів технічних засобів

Необхідна наявність IBM PC-сумісного ПК в комплекті і (і мережевого устаткування для режиму клієнт-сервер).

4.5.Вимоги до інформаційної та програмної сумісності

Дана програма повинна являти собою самостійний виконуваний модуль.

База даних повинна працювати під управлінням обраної (або заданої) СКБД (в розрахованому на багато користувачів режимі) в середовищі ОС (UNIX, Windows 95/98 або ін.). Програмне забезпечення (наприклад, інтерфейс) має бути написано на ... (одному з мов високого рівня ...) або (заданою мовою) або (мовою, вбудовану в СКБД).

(Якщо стоїть слово АБО, то вказується ЩОСЬ ОДНЕ !!)

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

В ході розробки програми має бути розроблено опис бази даних (схема БД) і підготовлена наступна документація:

- текст програми (або бібліотеки функцій);
- опис застосування;
- керівництво програміста (або керівництво користувача).

6. СТАДІЇ І ЕТАПИ РОЗРОБКИ

6.1. Ескізний проект

На цій стадії здійснюється вибір СКБД і іншого програмного забезпечення, визначається конфігурація технічних засобів.

Проводиться аналіз предметної області, за результатами якого детально розробляються структура АІС і бази даних із зазначенням кола вирішуваних завдань, обмежень цілісності і груп користувачів АІС. Розробляється загальний опис алгоритму, сам алгоритм.

Визначаються заходи з охорони праці, план заходів щодо створення АІС і бази даних. Розробляється пояснювальна записка.

Термін закінчення: "____" _____ 201 р.

6.2. Технічний проект

Створюється база даних під управлінням обраної СКБД .

Розробляються методи контролю вихідної інформації, засоби обробки помилок і видачі діагностичних повідомлень, запити до бази даних. Розробляється програмне забезпечення, визначаються методика і порядок його випробувань.

Вносяться доповнення в пояснювальну записку.

Термін закінчення: "_____" _____ 201 р.

6.3. Робочий проект

На цій стадії здійснюється налагодження програмного забезпечення. База даних заповнюється тестовими (або реальними) даними, готуються контрольно-налагоджувальні приклади, проводиться тестування АІС. Оцінюються тимчасові характеристики роботи системи, проводяться заходи, спрямовані на підвищення ефективності виконання критичних операцій (запитів).

Розробляються програмні документи. Остаточно оформлюються пояснювальна записка і графічний матеріал.

7. ПЕРЕЛІК ГРАФІЧНОГО МАТЕРІАЛУ

В ході розробки програми повинен бути в наявності такі графічний матеріал/ презентація (ВИБРАТИ ЗІ СПИСКУ ТЕ, ЩО ПІДХОДИТЬ САМЕ ДО ВАШОЇ РОБОТИ: має бути 6-7 листів):

1. Постановка завдання (0,5 аркуша).
2. Аналіз аналогів (або системного програмного забезпечення) - ЗАЗНАЧАЄТЬСЯ ЩОСЬ ОДНЕ !! (0,5 аркуша).
3. Вхідні і вихідні дані (0,5 аркуша).
4. Схема бази даних (1 аркуш).
5. Схема роботи програми (1 аркуш).
6. Приклад таблиць БД (1 аркуш).
7. Фрагменти роботи програми (1-2 листи).

8. ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контролює та прийом розробки здійснюються на основі випробувань контрольно-налагоджувальних прикладів. При цьому перевіряється виконання всіх функцій програми.

Термін здачі проекту: "____"_____ 201 р.